EDITED BY BILL TRAVIS & ANNE WATSON SWAGER

Software routine minimizes large logic tables

JERZY CHRZASZCZ, WARSAW UNIVERSITY OF TECHNOLOGY, WARSAW, POLAND

You can define combinatorial logic circuitry by explicitly stating the output patterns that result from particular input patterns. Such behavioral description is sometimes more convenient than writing logic equations; virtually all logic-design systems accept table entry, among other entry formats. Unfortunately, limitations of allowable table size preclude direct minimization of large logic tables.

The technique proposed here uses Espresso, a public-domain logic minimizer available from the University of California—Berkeley. You must decompose the table into fragments that fit Espresso limits. You then convert the fragments into Berkeley format. Espresso minimizes the fragments into smaller tables containing ones, zeros, and "don't cares." It then merges, reformats, and reprocesses the results.

LISTING 1—SAMPLE MINIMIZATION RESULTS

```
.i 14
.o 8
.p 16
-------------- 10000000
-------------- 01000000
-------------- 00100000
-1------------ 00010000
-1------------ 00001000
-------------0 00000100
-------0------ 00000100
------1------- 00000100
-0------------ 00000100
--1----------- 00000100
------1------- 00000010
--0----------- 00000010
-0------------ 00000010
--------1----- 00000001
------1------- 00000001
--0----------- 00000001
.e
```

LISTING 2—SAMPLE DOS BATCH PROGRAM

```
heco 14 8 0000 800 <xmpl.hex >q0
espresso <q0 >res
heco 14 8 0800 800 <xmpl.hex >q1
espresso <q1 >>res
heco 14 8 1000 800 <xmpl.hex >q2
espresso <q2 >>res
heco 14 8 1800 800 <xmpl.hex >q3
espresso <q3 >>res
heco 14 8 2000 800 <xmpl.hex >q4
espresso <q4 >>res
heco 14 8 2800 800 <xmpl.hex >q5
espresso <q5 >>res
heco 14 8 3000 800 <xmpl.hex >q6
espresso <q6 >>res
heco 14 8 3800 800 <xmpl.hex >q7
espresso <q7 >>res
glue 14 8 3 <res >res_
espresso -Dso <res_ >bits_
```

LISTING 3—HEX-TO-BERKELEY CONVERSION

```
/*---------- heco.c ------------*/
/* HECO - HEX->Espresso converter */
/* ver. 2.1              JCh 1997 */
/*----------------------------------*/

#include  <string.h>
#include  <stdio.h>

void   bin(unsigned int val, int pos)
{
    while (--pos >= 0)
    {
    if ((val>>pos) & 1)
      printf("1");
    else
      printf("0");
    }
    return;
}

int main(int argc, char *argv[])
{
    unsigned char  *str, flag = 0;
    unsigned int   cnt, adr, val;
    unsigned int   inputs = 16, outputs = 8;
    unsigned int   start = 0, bytes = 2048, stop;

    if (argc > 1)
    {
       argv[1] = strlwr(argv[1]);
       flag = ((strstr(argv[1],"?") != NULL)
       || (strstr(argv[1],"/h") != NULL)
       || (strstr(argv[1],"-h") != NULL));
    }
    if (argc == 1 || flag)
    {
       printf("\nHECO: Intel HEX -> espresso format converter (C) JCh 1997");
       printf("\nHECO  inp_cnt out_cnt [start_adr] [byte_cnt] <input_file >output_file\n");
       return 0;
    }
    sscanf(argv[1],"%2d",&inputs);
    if (argc > 2)
    {
       sscanf(argv[2],"%ld",&outputs);
       if (argc > 3)
       {
          sscanf(argv[3],"%4x",&start);
          if (argc > 4)
             sscanf(argv[4],"%4x",&bytes);
       }
    }
/*
:10 0020 00 E7E7E7E7E7E7E7E7E7E7E7E7E7E7E7E7 60
*/
    printf(".type fr\n.i %d\n.o %d\n",inputs,outputs);
    if (bytes == 0)
       stop = 0;
    else
    {
       stop = start + bytes - 1;
       printf(".p %d\n",bytes);
    }
    while (getchar() == ':')
    {
       scanf("%2x%4x%2s",&cnt,&adr,str);
       if (adr < start)
          gets(str), cnt = 0;
       else
          if (stop > 0  && adr > stop)
             break;
       while (cnt-- > 0)
       {
          scanf("%2x",&val);
          bin(adr++,inputs);
          printf(" ");
          bin(val,outputs);
          printf("\n");
       }
       gets(str);
    }
    puts(".e");
    return 0;
}
```

Group minimization occurs in the first pass, and minimization of each output function takes place in the second pass. The final table contains sets of vectors corresponding to product terms, which you can easily translate into logic equations. **Listing 1** shows the results obtained for a sample input table comprising 16,384 vectors of 14 inputs and eight outputs. **Listing 2**, a batch file, calls up the executable program heco.com for converting portions of the hex file xmpl.hex, which contains an input table (image of a 16-kbyte memory in hex format). The batch file also calls up heco.com, which converts portions of the hex file into Berkeley format, and glue.com, which reformats the merged results before the second processing pass. **Listings 3** and **4** show the source code for heco.com and glue.com, respectively. You can download the executable and source-code files from *EDN*'s Web site, www.ednmag.com. At the registered-user area, go into the Software Center to download the files from DISIG, #2042. (DI #2042) EDN

To Vote For This Design, Circle No. 437

LISTING 4—REFORMATTING MERGED RESULTS

```
/*---------- glue.c ------------*/
/* GLUE - merged Espresso gluer   */
/* ver. 2.2              JCh 1997 */
/*---------------------------------*/

#include  <string.h>
#include  <stdio.h>


void   bin(unsigned int val, int pos)
{
    while (--pos >= 0)
    {
    if ((val>>pos) & 1)
       printf("1");
    else
       printf("0");
    }
    return;
}


int main(int argc, char *argv[])
{
    char  c, *str, *adr, *val, *format, flag = 0;
    unsigned int   inputs = 16, outputs = 8;
    unsigned int   bits = 4, count = 0;

    if (argc > 1)
    {
       argv[1] = strlwr(argv[1]);
       flag = ((strstr(argv[1],"?") != NULL)
       || (strstr(argv[1],"/h") != NULL)
       || (strstr(argv[1],"-h") != NULL));
    }
    if (argc == 1 || flag)
    {
       printf("\nGLUE: merged espresso -> espresso format converter (C) JCh 1997");
       printf("\nGLUE  inp_cnt out_cnt bit_cnt <input_file >output_file\n");
       return 0;
    }
    sscanf(argv[1],"%2d",&inputs);
    if (argc > 2)
    {
       sscanf(argv[2],"%1d",&outputs);
       if (argc > 3)
          sscanf(argv[3],"%d",&bits);
    }
    printf(".i %d\n.o %d\n",inputs,outputs);
    while ((c = getchar()) != EOF)
    {
       gets(str);
       if (c == '.' && str[0] == 'e')
          count++;
       if (c == '-')
       {
          bin(count,bits);
          puts(&str[bits-1]);
       }
    }
    puts(".e");
    return 0;
}
```

Spice subcircuit models thermistors

LUTZ WANGENHEIM, HOCHSCHULE, BREMEN, GERMANY

The subcircuit in **Figure 1** is a simple and efficient behavioral model for a thermistor, implemented in PSpice (Microsim, Irvine, CA). The model simulates realistic thermistor parameters for all standard analyses (transient, ac, and dc). You can set all relevant thermistor parameters—nominal resistance, nominal and ambient temperatures, and material and thermal constants—during the call of the subcircuit. **Listing 1** gives the parameter definitions for the subcircuit. The model reflects **Equation 1**, traditionally used to describe the resistance-temperature characteristic of thermistors:

$$R_T = R_{NOM} \bullet \exp(B/T_B - B/T_{NOM}), \qquad (1)$$

where R_T is the resistance at thermistor-body temperature T_B, R_{NOM} is the nominal resistance at nominal (standard reference) temperature T_{NOM}, and B is the material constant (in Kelvin) that describes the temperature sensitivity.

Note that **Equation 1** is valid only within a limited temperature range, because it does not reflect the actual temperature dependence of the material constant, B. However, with respect to manufacturing tolerances of B, the **equation** provides a useful approximation. The subcircuit establishes thermal feedback by using a conventional ohmic resistor, representing R_{NOM}, in series with a voltage source E_{TEMP} that represents the temperature-dependent term. The independent zero-voltage source, V_{SENSE}, senses the current, I_T, through the circuit.

The voltage, $V_T=I_TR_T$, generated between thermistor nodes 1 and 2 is the sum of the voltage across R_{NOM} and the output, V_{TEMP}, of the voltage source, E_{TEMP}:

$$I_T \bullet R_T = I_T \bullet R_{NOM} + V_{TEMP}.$$

Replacing R_T by **Equation 1** and solving for V_{TEMP} yields the control function for E_{TEMP}. You can derive the function by applying the analog-behavioral model of PSpice in **Listing 1**:

$$V_{TEMP} = I_T \bullet R_{NOM}\left[\exp(B/T_B - B/T_{NOM}) - 1\right].$$

Because all control parameters of the dependent source must be constants, currents, or voltages, you need some extra circuitry to compute the body temperature, T_B, and convert it to a corresponding voltage. Therefore, you use a dependent current source, G_{PWR}, controlled by the voltage-current product, $V_T I_T$, to generate a voltage across resistor R_{TH} at Node 5. This voltage represents the power-dependent portion of the body temperature, T_B, if you set the value of the thermal resistance, R_{TH}, to the reciprocal of the thermistor's dissipation factor, D. Capacitor C_{TH} in parallel with R_{TH} models the thermal time constant, τ, of the thermistor body.

For ambient temperature-to-voltage conversion, a constant current, I_{AMB}=1A, must produce a voltage at Node 6 that is numerically equal to the ambient temperature, T_{AMB}, in Kelvin. This temperature constitutes the second portion of T_B. To satisfy this equivalency, you use a resistor model for R_{AMB} with a nominal value of 300.15Ω and a linear temperature coefficient of TC=1/300.15=3.33×10$_3$. During simulation runs, you can set the ambient temperature (for PSpice, nominally 27°C, or 300.15K) with a global .TEMP statement.

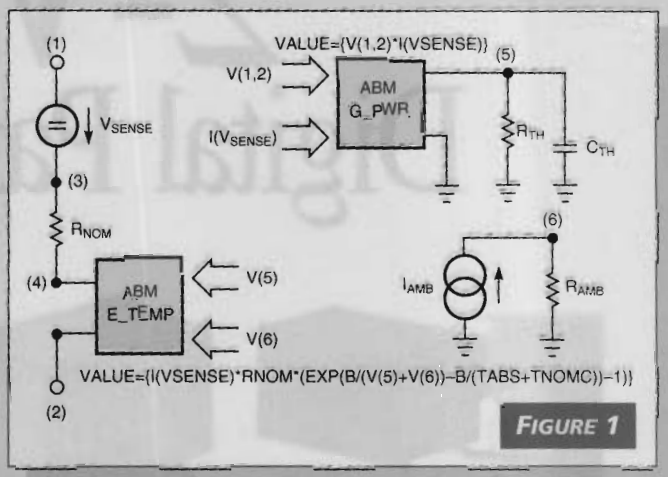


FIGURE 1

This thermistor model incorporates thermal feedback to accurately model current-voltage characteristics for transient analysis as well as for dc and ac simulations.

Note that ac simulation, which is a linear (small-signal) analysis, must not cause any change in the thermistor's electrical resistance. Therefore, the voltage at Node 5, a measure of internal heating, must not contain any ac portion that may result from the nonlinear control operation (current-voltage multiplication) of the behavioral-modeling block, G_{PWR}. Unfortunately, if any dc flows through the thermistor during bias-point calculation (before the ac analysis), the output of G_{PWR} will always contain such an unwanted ac portion. Therefore, to make the ac voltage at Node 5 negligible, you should increase the value of C_{TH} accordingly.

During ac simulations, you should set the time constant, $\tau=R_{TH}C_{TH}$, to a much larger value than $1/f_{MIN}$, the start frequency of the ac analysis. Thus, the current-voltage characteristic of the thermistor becomes a function of only its nominal value, R_{NOM}; the ambient temperature, T_{AMB}; and the dc bias flowing through the device. You can download the PSpice file from *EDN*'s Web site, www.ednmag.com. At the registered-user area, go into the Software Center to download the files from DI-SIG, #2043. (DI #2043) EDN

To Vote For This Design, Circle No. 438

LISTING 1—SUBCIRCUIT DESCRIPTION FOR THERMISTOR MACRO MODEL

```
*LISTING MACRO MODEL "THERMISTOR"
.SUBCKT THERM 1 2
+ PARAMS: TABS=273.15  TNOMC=25  RNOM=1k  B=3000  D=1E-3  TAU=1
*
*DEFAULT SETTINGS TO BE MODIFIED DURING SUBCIRCUIT CALL
*TNOMC=STANDARD REF. TEMPERATURE IN DEG. CELSIUS
*RNOM=NOMINAL RESISTANCE VALUE AT TNOMC
*B=THERMISTOR CONSTANT IN K
*D=THERMISTOR DISSIPATION FACTOR IN W/K
*TAU=THERMAL TIME CONSTANT IN SEC.
*>>> HINT FOR AC ANALYSIS:  SET TAU>100/FMIN <<<
*********************************************
* BASIC THERMISTOR MODEL
*
VSENSE  1 3 DC 0V
RNOM    3 4 {RNOM}
E_TEMP 4 2 VALUE=
+{I(VSENSE)*RNOM*(EXP(B/(V(5)+V(6))-B/(TABS+TNOMC))-1)}
*********************************************
*POWER-TO-TEMPERATURE-TO-VOLTAGE CONVERSION
*
G_PWR 0 5 VALUE={V(1,2)*I(VSENSE)}
RTH     5 0 {1/D}
CTH     5 0 {TAU*D}
*********************************************
*AMBIENT TEMPERATURE-TO-VOLTAGE CONVERSION
*
IAMB  0 6 1A
RAMB  6 0 300.15  TC=3.33E-3
.ENDS THERM
*********
```

Battery monitor emulates auto dashboard

TG BARNETT AND J MILLAR, UNIVERSITY OF LONDON, UK

When you use battery-powered scientific-recording equipment, you must know the state of the battery, because an unnoticed battery run-down can lead to loss of data. Several types of warning indicators are available, but all have their drawbacks. Moving-coil indicators, test terminals, and digital flags, for example, all are wanting in some respect—they can be inaccurate or unreliable, have an unacceptable battery drain, or have some other operational problem. Ideally,

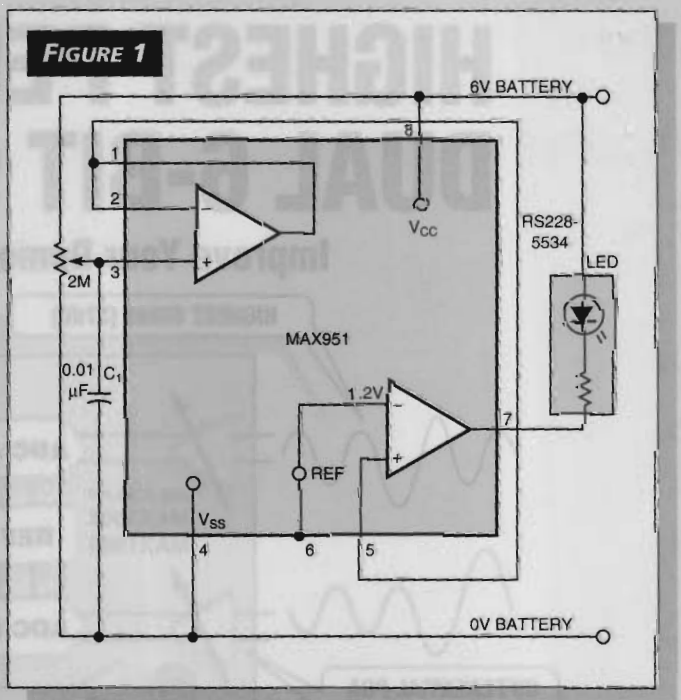
you need a passive and fail-safe system requiring no action on your part. However, if anything goes wrong, you should receive an automatic alert. The automobile industry inspired the simple circuit in **Figure 1**.

When you start the ignition in a modern vehicle, the warning lights on the dashboard briefly come on and then go out if all is well. The same thing happens with the configuration in **Figure 1**: Upon equipment switch-on, the super-bright LED momentarily lights up, thereby confirming an adequate power-supply voltage. The LED then goes out, but if the battery voltage falls below a set level during operation, it permanently switches on. The circuit uses a MAX951 ultralow-power, single-supply op amp/comparator/reference, configured as a unity-gain comparator.

The 2-MΩ potentiometer sets the voltage at the positive input of the op amp; in this example, at 1.44V. When the battery voltage falls to 5V, the comparator output goes low, thus illuminating the LED. At switch-on, with a 0.01-μF value for C_1, the LED gives an approximately 30-msec flash. Note that the MAX951 can operate with a supply voltage as low as 2.4V. (DI #2052) EDN

To Vote For This Design, Circle No. 439

FIGURE 1



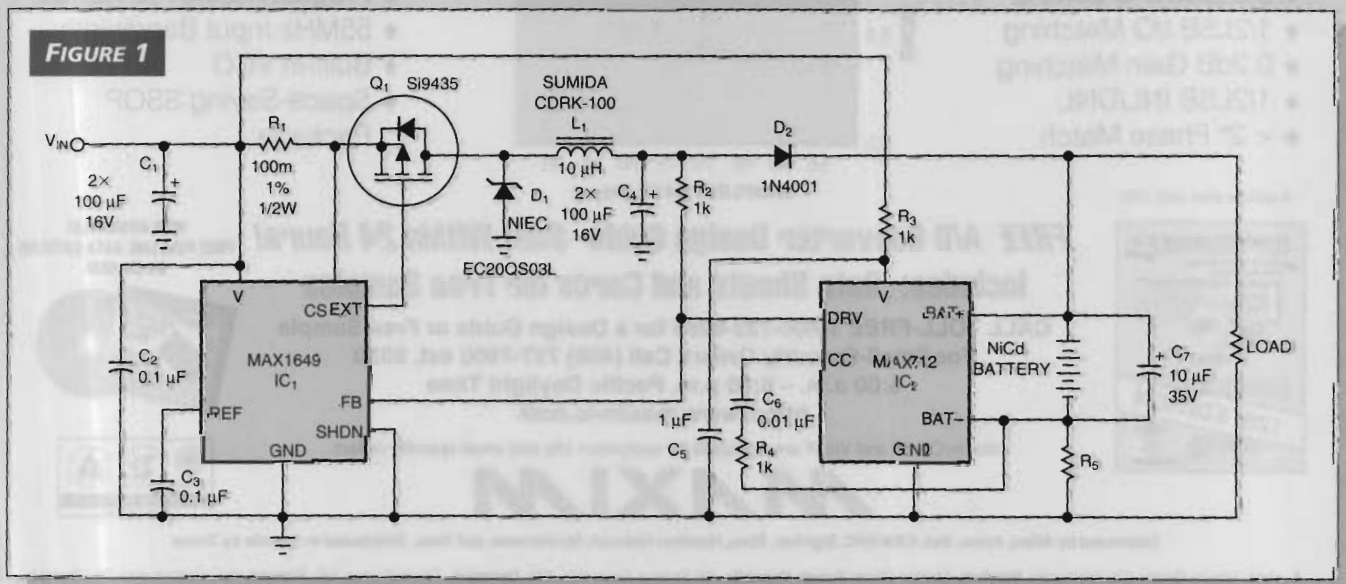
Similar to your car's dashboard lights, the LED in this battery monitor momentarily flashes upon power-up, then goes out. If the battery voltage is low, the LED stays lit continuously.

Charger serves load while charging battery

BERNARD KONRAD, MAXIM INTEGRATED PRODUCTS, GRAEFELFING, GERMANY

The battery-charger circuit in **Figure 1** monitors the battery voltage and automatically initiates a recharge if required. Its output-current capacity is sufficient to simultaneously charge the battery and supply the load. It thus provides continuous power to the load during recharge operations. The circuitry in IC_2 provides "smart" charge terminations for either NiCd or NiMH batteries. Unlike chargers that use a pulsed trickle charge during power standby, this circuit does not disconnect the load from the power source. The battery and load remain connected in parallel, so the battery is always fully charged when you remove power. Other advantages include high efficiency, small circuit area, and the use

FIGURE 1



A battery charger controls a switch-mode current source, which supplies the battery and load during both standby and power-surge conditions.

of inexpensive standard components.

IC_1 and its associated components form a low-dropout, step-down dc-dc controller configured as a switch-mode current source. IC_2 is a fast-charge controller for batteries. Connecting IC_2's DRV output, which normally drives the base of a pnp transistor's linear current source, to IC_1's feedback terminal causes IC_2's current-regulator circuitry to control the external switch-mode current source. Connecting pullup resistor R_2 to the battery voltage via D_2 provides negative voltage feedback that improves the overall dynamic performance. R_5's value is a function of the desired charge rate.

To enable the circuit to deliver a controlled fast charge as well as a regulated low trickle charge, IC_2's CC input (compensation for the chip's constant-current regulation loop) terminates in the R_4-C_6 series RC network. The network integrates the internal error signal for the loop. Load current and battery-charging current come from the same source, with the battery acting as a buffer. If a load surge, such as the high turn-on current for a printer motor, depresses the battery voltage and causes the charger to fall out of regulation, the circuit applies a fast charge until IC_2 senses the full-charge condition. The circuit then resumes its regulated trickle-charge mode. (DI #2044) EDN

To Vote For This Design, Circle No. 440

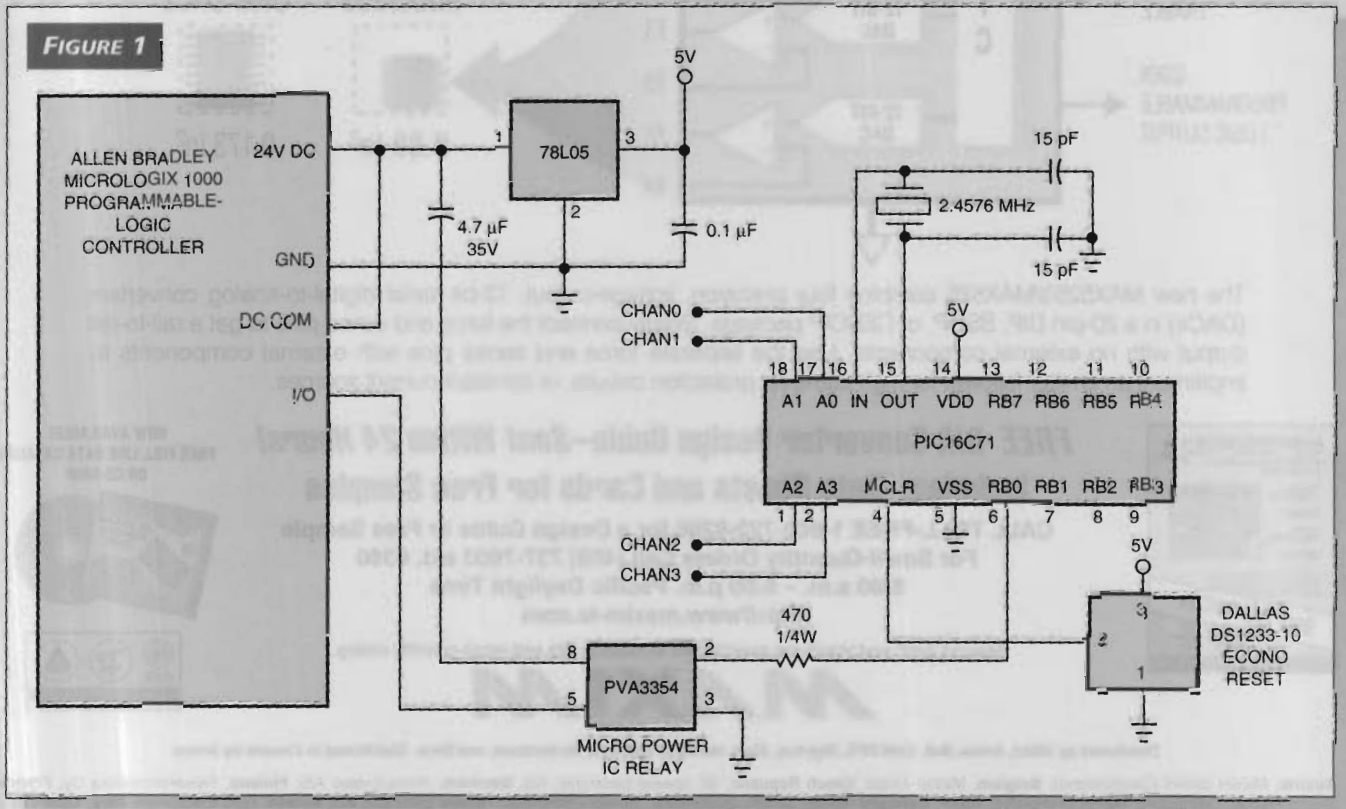
µC provides analog data to PLC chip

LUKE BARKER, REINKE MFG CO, DESHLER, NE

The low price of programmable-logic controllers (PLCs) makes these small devices a good choice for many system designs. Most of the controllers, however, accommodate only digital inputs. One such model is Allen Bradley's (Milwaukee, WI) MicroLogix 1000 PLC, which has 10 digital (on/off) inputs and six relay or solid-state outputs. In certain applications, the need exists to provide analog inputs to the PLC. The circuit in Figure 1 sends analog data serially to a PLC, using only one digital input on the PLC. The Microchip PIC16C71 µP provides A/D conversion and sends four converted channels to the PLC. The µP sends the data to the PLC using only one output pin on the µP and one input port on the PLC. (Note that the PLC must be the 24V-dc-input version.)

The µP receives its power from the 24V power supply (200

FIGURE 1



An inexpensive µP and a handful of components make it possible to provide analog data to a digital-only programmable-logic controller (PLC).

mA maximum) and a 5V regulator. The chip sends 12 bits of information to an optoisolated IC relay. The relay, in turn, translates the 5V pulses to 24V pulses for the PLC's input. The data consists of 1 start bit, 2 channel-ID bits, 8 bits corresponding to the analog value, and 1 stop bit. In receiving the data, the PLC uses its selectable-timed-interrupt feature. The fastest interrupt time is 10 msec, but here you use 20 msec to minimize interrupt latency. The PLC accepts 48 data bits (12 bits, four channels) to receive all four channels of analog data. Thus, it takes approximately 0.96 sec to receive all four channels, corresponding to a baud rate of 50 bps. If the data stops transmitting for some reason, the PLC simply stops interrupting until it detects a start bit again; normal operation then continues.

The 2.4576-MHz crystal oscillator provides accurate 20-msec time-delay intervals that the PLC needs for synchronization. The start bit is 30 msec long, and each data bit is 20 msec long. The longer start bit allows the PLC to recognize the beginning of a start bit, perform an interrupt after 20 msec, and systematically interrupt every 20 msec thereafter. The synchronization is such that the PLC interrupts at the center of each data bit. (You can download the PIC16C71 Parallax-compiler code, as well as a graphical representation of the PLC source listing, from *EDN*'s Web site, www.ednmag.com. At the registered-user area, go into the Software Center to download the file from DI-SIG, #2051.) (DI #2051)

To Vote For This Design, Circle No. 441

Spice model simulates spark-gap arrestor

CHRISTOPHE BASSO, SINARD, FRANCE

Spark-gap arrestors, or surge arrestors, are highly nonlinear devices whose function is to block transient surges on dc or ac power-supply lines. Such transients can arise from lightning strikes, motor starts, and other causes. A spark gap uses two electrodes that oppose each other across a short distance in an atmosphere of inert gas, such as neon or argon. If the voltage on the arrestor is below its striking voltage (avalanche potential), the current in the arrestor is close to zero. Once the potential reaches the striking voltage, the voltage across the arrestor suddenly collapses to a level called the glow voltage. If the current increases further, the arrestor voltage decreases to a level called the arc voltage, where it stays until the surge passes. At this point, the arrestor stays conductive until its current falls below a sustaining value, much in the manner of a thyristor. Because the gas needs a certain amount of time for ionization, the ignition voltage depends on the dv/dt applied to the arrestor. The Spice model presented here takes into account the dv/dt effect.

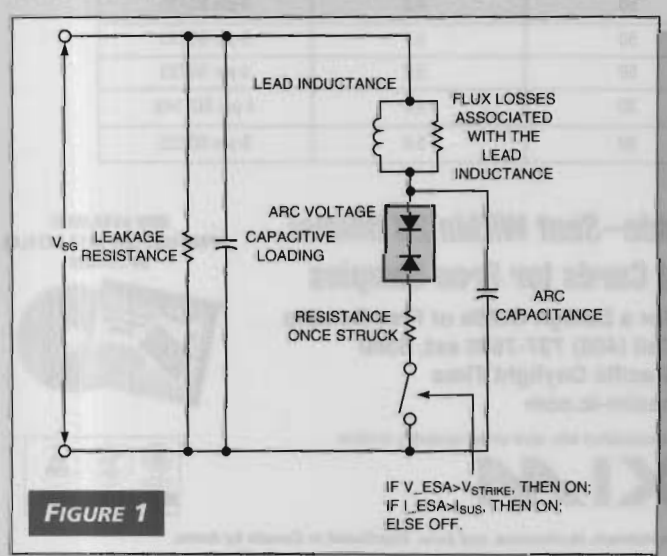


FIGURE 1

Back-to-back zener diodes and a voltage-controlled switch form the heart of a circuit that simulates a spark-gap surge arrestor.

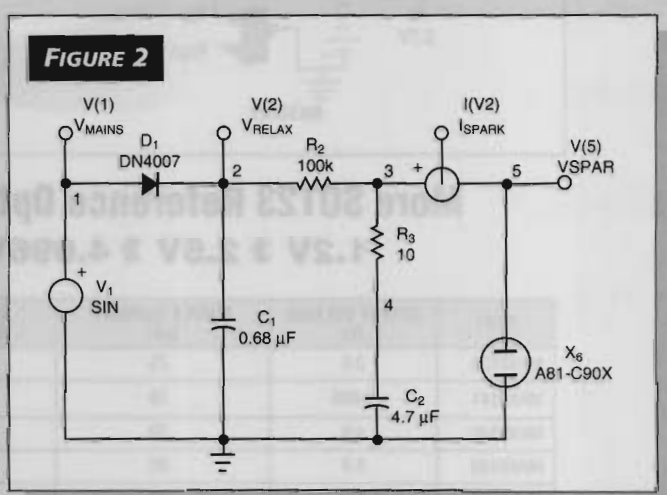


FIGURE 2

A self-relaxing configuration replaces the Siemens A81-C90X arrestor with the model in Figure 1 for the Spice simulation.

You can model such a component with Spice in several ways (**References 1** and **2**). For simplicity and easy implementation, the method shown uses the macromodeling technique. The model assembles Spice primitives to describe a complex electrical function. **Figure 1** shows the general spark-gap model. In the off state, the voltage-controlled switch is open, and only a leakage current flows in the spark gap. The switch stays off until V_{SG} increases to the striking voltage, V_{STRIKE}. At this point, the switch immediately turns on and the network comprising the back-to-back zener diodes and the series resistance connects across the arrestor's

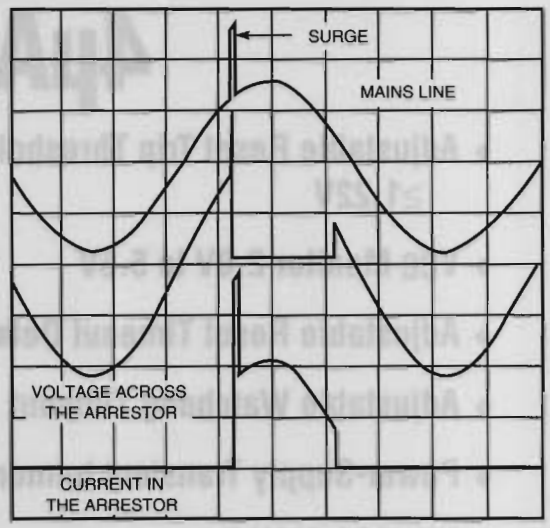
terminals. The voltage then collapses to the arc value, and the current starts to rise.

When the surge passes, the arrestor current decays until the voltage reaches the sustaining value, and the switch opens. This simple model does not take into account the glow transition. The netlist in **Listing 1** uses IntuSoft's (San Pedro, CA) IsSpice 4. (You can download **Listing 1** from *EDN*'s Web site, www.ednmag.com. At the registered-user area, go into the Software Center to download file DI2053.lst from DI-SIG.) IsSpice 4 uses standard Spice 3 elements combined with one of IntuSoft's Spice extensions: an if-then-else behavioral element (BARC). The A1 element in the Spice routine depicts, point by point, the way the ignition voltage increases in the presence of higher voltage slopes. You can easily obtain this ignition-voltage information by observing the $V_{IGNITION}$/dv/dt curves in the arrestor's data sheet. A classic differentiator calculates the slope of the input signal.

To adapt the model to a particular spark-gap device, you need only enter the parameters in the data sheet. The default values shown correspond to a Siemens (Iselin, NJ) A81-C90X surge arrestor. The first Spice test uses the self-relaxing configuration in **Figure 2**. Because the surge phenomena are fast, you need to view the raw, noninterpolated Spice data and not the interpolated (.PRINT) data. Using IntuSoft's IntuScope graphical-investigation tool, you can easily explore both types of data. **Figure 3** shows the Spice results. Oscilloscope measurements on an actual circuit show close correlation with the Spice simulation.

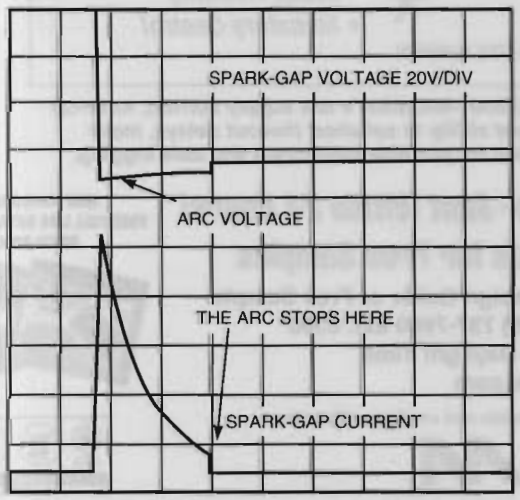
A second test uses the spark-gap device as a real surge arrestor. The power mains supply a device protected by an arrestor. A 1-µsec transient, added to the supply voltage, sets off the arrestor. **Figure 4** shows the Spice-simulated results. You simulate the dv/dt effects by driving the arrestor with different slopes. As the slope increases, so does the ignition voltage. **Figure 5** shows the resulting curves. This model

FIGURE 4



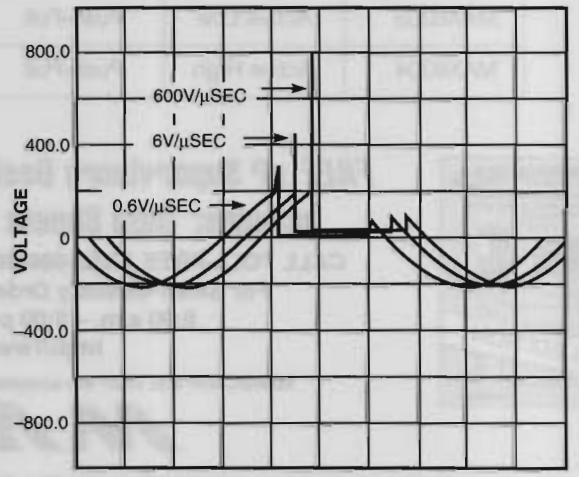
Spice results show that a surge in the line voltage provokes clamping at the arcing voltage; the normal line voltage returns after the current falls below the arc-sustaining value.

FIGURE 3



The Spice simulation of the self-relaxing configuration in Figure 2 shows the beginning and end of the spark-gap arcing region.

FIGURE 5



The ignition voltage of a spark-gap surge arrestor is a direct function of the dv/dt slope of the surge signal.